IN THE SPECIFICATION

Please replace the paragraph beginning on page 10, line 19 with the following replacement paragraphs:

Because of a the wire-line-level protocol, each token must be processed sequentially starting from token_1 through token_4. Each token is a sample of an external bus (not illustrated). The tokens are demultiplexed from the external bus and registered in an input register 225. Input register 225, in conjunction with other circuitry in FPGA 300, is clocked using an internal clock 280 that cycles at 1/4th the rate of the external bus' clock rate. Thus, finite state machine 210 must sequentially process token_1 through token_4 in a single cycle of internal clock 280. The state machine processing for each token is the same so that the memory contents for each ROM 201 through 204 are identical and are easily obtainable based upon the required state diagram for finite state machine 210 ~~220~~. In turn, the required state diagram depends upon the particular wire-line-level protocol being implemented.

A state diagram for a SPI4 protocol is shown in Figure 3. There are three possible states: an idle state (idle), a payload control state (pld), and a data state (data). The corresponding input conditions for this state diagram are derived from the tokens (or from their associated out-of-band signals). These input conditions are denoted as idle, payload control (pctl), and rctl. These input conditions are mutually exclusive in that only one of them should be active at any given time. Violations of this mutual exclusivity result in error flag production. For example, if rctl goes active in the idle state while idle is active, an idle error flag is generated. It will be appreciated, however, that the specifics of the state machine being implemented are unimportant to the present invention.

Please replace the paragraph beginning on page 12, line 1 with the following replacement paragraph:

Each ROM 201 through 204 is programmed to do the finite state machine sequencing for its corresponding token, token_1 through token_4, respectively. Thus, ROM 201 is programmed to determine the next state for finite state machine ~~220~~ 210

LAW OFFICES OF
MACPHERSON, KWOK CHEN
& HEID LLP

2402 MICHELSON DRIVE
SUITE 210
IRVINE, CA 92612
(949) 752-7040
FAX (949) 752-7049

based upon input conditions determined from token_1, ROM 202 does the same for token_2, ROM 203 for token_3, and ROM 204 for token_4. ROMs 201 through 204 are each programmed to provide the next state for finite state machine ~~220~~ 210 based upon the current state and input conditions from the corresponding token, token_1 through token_4, respectively. ROMs 201 through 204 are stacked such that data 230 from ROM 201 become address inputs for ROM 202. Thus, ROM 201 is programmed such that data outputs 230 form the next state for finite state machine ~~220~~ 210. Additional address inputs 235 from token_2 to ROM 202 supply the current input conditions. ROM 202 is thus programmed to determine the next state for finite state machine ~~220~~ 210 (data outputs 240) based upon the current state (address inputs 230) and the current input conditions (address inputs 235). In turn, data outputs 240 as used by ROM 203 as address inputs 240 in conjunction with additional address inputs 245 from token_3 to select a data output 250 from ROM 203 ~~204~~. Using data output 250 as an address input (to provide the current state) in conjunction with additional address inputs 255 from token_4 (to provide the current input conditions), ROM 204 retrieves a data output 260 (to specify the next state). Because data output 260 will serve as address inputs 260 for ROM 201 in the next cycle of internal clock 280, data output 260 must be registered. Although a state register 205 is shown in Figure 2 to perform this registration, it is symbolic in that each programmable block 305 of FPGA 300 may be configured to register its outputs within its macrocells (not illustrated). Accordingly, no routing need be performed to accomplish the registration of data output 260. In the subsequent cycle of clock 280, ROM 201 uses data output 260 as an address input 260 in conjunction with address inputs 275 derived from token_1. Unlike data output 260 from ROM 204 ~~205~~, the data outputs from ROMs 201 through 203 need not be clocked or registered.

Please replace the paragraph beginning on page 14, line 12 with the following replacement paragraph:

The finite state machine 210 ~~220~~ of Figure 2 may be generalized to process any number of tokens in a given cycle of internal clock 280 ~~270~~ and instantiated into FPGA 300. To perform this instantiation, a programmable block 305 is assigned to each token that must be processed. The tokens are arranged from a first-to-be analyzed token to a

LAW OFFICES OF
MACPHERSON, KWOK CHEN
& HEID LLP

2402 MICHELSON DRIVE
SUITE 210
IRVINE, CA 92612
(949) 752-7040
FAX (949) 752-7049

last-to-be analyzed token. Thus, a first programmable block corresponds to the first-to-be analyzed token, a second programmable block corresponds to the next-to-be analyzed token, and so on until a final programmable block corresponds to the last-to-be analyzed token. The data outputs of the final programmable block are clocked and fed back as address inputs to the first programmable block. The remaining programmable blocks are daisy-chained as described with respect to Figure 2 such that the data outputs of the first programmable block become the address inputs to the second programmable block, and so on. Each programmable block 305 from the first to the last are instantiated as a ROM programmed to calculate the next state of finite state machine 210 ~~220~~ based upon the current state and the current input conditions. For a given ROM, the current state corresponds to the next state determined by the preceding ROM. The current input conditions are derived from each ROM's corresponding token. In this fashion, finite state machine 210 ~~220~~ is sequenced from state-to-state as it processes each token.

Please replace the paragraph beginning on page 14, line 12 with the following replacement paragraph:

As is known in the art, a variety of synthesis software tools may be used to instantiate finite state machine 210 ~~220~~ into a programmable device such as FPGA 300. For example, the following section of RTL code may be used to capture a 4-level (to process 4 tokens) finite state machine. The SPI4 wire-line-level protocol for the 4 tokens is such that the finite state machine may be in one of four states in response to each token. Thus, 2 bits are necessary to specify the state. Four ROMS are instantiated into the programmable device, one for each token. Each ROM may have a depth of 32 words such that 5 address bits are necessary to retrieve a particular data word. Each retrieved data word determines the next state of the desired finite state machine. From the retrieved data word, two bits such as the least two significant bits may be used to specify the next state as address bits for the next ROM stage. Additional bits in the retrieved data word may be used for control and other functions unrelated to the necessary state sequencing. From each token, the current input conditions are derived as three additional address bits. From the following example RTL code segment, the programmable blocks will be instantiated into ROMs such that the outputs of one ROM may form the inputs to

LAW OFFICES OF
MACPHERSON, KWOK CHEN
& HEID LLP

2402 MICHELSON DRIVE
SUITE 210
IRVINE, CA 92612
(949) 752-7040
FAX (949) 752-7049

an adjacent ROM as described previously. For example, the first ROM is chosen to have the coordinates "R28C9" indicating the row (number 28) and the column (number 9) position of the programmable block that will be instantiated to form the first ROM. The other ROMs are formed in the same row such that the second ROM is in column 10, the third ROM is in column 11, and the fourth ROM is in column 12. Note also that the content of each ROM is easily conveyed as follows:

```
/*
 *
 * Address Inputs       Data Outputs   Programming
 *
 *
 *  A4 A3 A2 A[1:0]  |  D3 D2  D[1:0]    |
 *       inp_state   |      out_state    | Addr   Data
 * rctl pctl idle fsm_v[1:0]| fsm_v[1:0]   |
 * ------------------------|---------|--------------
 *  0   0  1  0x0 (Idle)   | 0 0 0 (Idle) | 0x4 0x0
 *  0   1  0  0x0 (Idle)   | 0 0 1 (pld)  | 0x8 0x1
 *  1   0  0  0x0 (Idle)   | 0 1 0 (Idle) | 0x10 0x4
 *
 *  0   0  1  0x1 (Pld)    | 1 0 0 (Idle) | 0x5  0x8
 *  0   1  0  0x1 (Pld)    | 1 0 1 (Pld)  | 0x9  0x9
 *  1   0  0  0x1 (Pld)    | 0 0 2 (Data) | 0x11 0x2
 *
 *  0   0  1  0x2 (Data)   | 0 0 0 (Idle) | 0x6  0x0
 *  0   1  0  0x2 (Data)   | 0 0 1 (Pld)  | 0xa  0x1
 *  1   0  0  0x2 (Data)   | 0 0 2 (Data) | 0x12 0x2
 *
`timescale 1 ns / 100 ps
module dps_fsm_rom (clk, idle_4b, pctl_4b, rctl_4b, idle_err, ctl_dat_err) ;


// Inputs
input clk;                      / FSM clock
```

LAW OFFICES OF
MACPHERSON, KWOK CHEN
& HEID LLP

2402 MICHELSON DRIVE
SUITE 210
IRVINE, CA 92612
(949) 752-7040
FAX (949) 752-7049

```
input [3:0] idle_4b;    / Idle bit array(Idle=1)
input [3:0]pctl_4b; /Payload control bitarray(pctl=1) input [3:0] rctl_4b; /Receive
Control word bit array (ctl=1, data=0)

// Outputs
   output idle_err; // Protocol error while idle
   output ctl_dat_err;          // Protocol error while in the payload or data state

// Wires and Regs
 wire [1:0] fsm0_v; // FSM intermediate state variables
 wire [1:0] fsm1_v;// FSM intermediate state variables
 wire [1:0] fsm2_v;     // FSM intermediate state variables
 wire [1:0] fsm3_v; // FSM intermediate state variables
 wire [3:0] idle_e;        // Idle error per FSM level
 wire [3:0] ctl_dat_e;// ctl/data error per FSM level
 reg  idle_err, ctl_dat_err;

// Collapse 4 level errors into single error bits
   always @(posedge clk) begin idle_err = |idle_e; end
   always @(posedge clk) begin ctl_dat_err =    |ctl_dat_e; end
//    always @(posedge clk) begin fsm0_vl <= fsm0_v ; end

/*
* Begin FSM processing: Note that processing occurs *left to right -> MSB is first
* received in time so start with most significant "_4b" * arrays slice [4]
*/

/* First Logic Level */
   ROM32X4 mem_3 (.AD0(fsm0_v[0]), .AD1(fsm0_v[1]), .AD2(idle_4b[3]),
                  .AD3(pctl_4b[3]), .AD4(~rctl_4b[3]), .CK(clk),
```

LAW OFFICES OF
MACPHERSON, KWOK CHEN
& HEID LLP

2402 MICHELSON DRIVE
SUITE 210
IRVINE, CA 92612
(949) 752-7040
FAX (949) 752-7049

```
                .DO0(fsm3_v[0]), .DO1(fsm3_v[1]), .DO2(idle_e[0]),
.DO3(ctl_dat_e[0]))
            /* synthesis initval="0x00000000000002240000019100800000" */
            /* synthesis comp="mem_3" *//* synthesis loc="R28C9" */;

/* Second Logic Level */
  ROM32X4 mem_2 (.AD0(fsm3_v[0]), .AD1(fsm3_v[1]), .AD2(idle_4b[2]),
                .AD3(pctl_4b[2]), .AD4(~rctl_4b[2]), .CK(clk),
                .DO0(fsm2_v[0]), .DO1(fsm2_v[1]), .DO2(idle_e[1]),
.DO3(ctl_dat_e[1]))
            /* synthesis initval="0x00000000000002240000019100800000" */
            /* synthesis comp="mem_2" *//* synthesis loc="R28C10" */;

/* Third Logic Level */
  ROM32X4 mem_1 (.AD0(fsm2_v[0]), .AD1(fsm2_v[1]), .AD2(idle_4b[1]),
                .AD3(pctl_4b[1]), .AD4(~rctl_4b[1]), .CK(clk),
                .DO0(fsm1_v[0]), .DO1(fsm1_v[1]), .DO2(idle_e[2]),
.DO3(ctl_dat_e[2]))
            /* synthesis initval="0x00000000000002240000019100800000" */
            /* synthesis comp="mem_1" *//* synthesis loc="R28C11" */;

/* Fourth Logic Level */
  ROM32X4 mem_0 (.AD0(fsm1_v[0]), .AD1(fsm1_v[1]), .AD2(idle_4b[0]),
                .AD3(pctl_4b[0]), .AD4(~rctl_4b[0]), .CK(clk),
                .QDO0(fsm0_v[0]), .QDO1(fsm0_v[1]), .QDO2(idle_e[3]),
.QDO3(ctl_dat_e[3]))
            /* synthesis initval="0x00000000000002240000019100800000" */
            /* synthesis comp="mem_0" *//* synthesis loc="R28C12" */;


endmodule
```

LAW OFFICES OF
MACPHERSON, KWOK CHEN
& HEID LLP

2402 MICHELSON DRIVE
SUITE 210
IRVINE, CA 92612
(949) 752-7040
FAX (949) 752-7049

IN THE DRAWINGS

Please replace the second and third sheet of formal drawings with the enclosed replacement sheets.

LAW OFFICES OF
MACPHERSON, KWOK CHEN
& HEID LLP

2402 MICHELSON DRIVE
SUITE 210
IRVINE, CA 92612
(949) 752-7040
FAX (949) 752-7049